CP/M DYNAMIC DEBUGGING TOOL (DDT)

USER'S GUIDE

Table of Contents

CP/M Dynamic Debugging Tool (DDT)

User's Guide

I. Introduction

The DDT program allows dynamic interactive testing and debugging of programs generated in the CP/M environment. The debugger is initiated by typing one of the following commands at the CP/M Console Command level:

```
DDT
DDT filename.HEX
DDT filename.COM
```

where "filename" is the name of the program to be loaded and tested. In both cases, the DDT program is brought into main memory in the place of the Console Command Processor (refer to the CP/M Interface Guide for standard memory organization), and thus resides directly below the Basic Disk Operating System portion of CP/M. The BDOS starting address, which is located in the address field of the JMP instruction at location 5H, is altered to reflect the reduced Transient Program Area size.

The second and third forms of the DDT command shown above perform the same actions as the first, except there is a subsequent automatic load of the specified HEX or COM file. The action is identical to the sequence of commands

```
DDT
Ifilename.HEX or Ifilename.COM
R
```

where the I and R commands set up and read the specified program to test (see the explanation of the I and R commands below for exact details).

Upon initiation, DDT prints the following sign-on message.

```
DDT VERS n.n
```

where nn represents the version number.

		the source file
p2:	A,B, ..., Y	designates the disk name which will receive the hex file
	Z	skips the generation of the hex file
p3:	A,B, ..., Y	designates the disk name which will receive the print file
	X	places the listing at the console
	Z	skips generation of the print file

Thus, the command

```
ASM X.AAA
```

indicates that the source file (X.ASM) is to be taken from disk A, and that the hex (X.HEX) and print (X.PRN) files are to be created also on disk A. This form of the command is implied if the assembler is run from disk A. That is, given that the operator is currently addressing disk A, the above command is equivalent to

```
ASM X
```

The command

```
ASM X.ABX
```

indicates that the source file is to be taken from disk A, the hex file is placed on disk B, and the listing file is to be sent to the console. The command

```
ASM X.BZZ
```

takes the source file from disk B, and skips the generation of the hex and print files (this command is useful for fast execution of the assembler to check program syntax).

The source program format is compatible with both the Intel 8080 assembler (macros are not currently implemented in the CP/M assembler, however), as well as the Processor Technology Software Package #1 assembler. That is, the CP/M assembler accepts source programs written in either format. There are certain extensions in the CP/M assembler which make it somewhat easier to use. These extensions are described below.

2. PROGRAM FORMAT.

An assembly language program acceptable as input to the assembler consists of a sequence of statements of the form

```
line#    label    operation    operand    ;comment
```

where any or all of the fields may be present in a particular instance. Each

which reloads previously saved program from loaction 100H through page 18 (12FFH). The machine state is not a part of the COM file, and thus the program must be restarted from the beginning in order to properly test it.

II. DDT COMMANDS.

The individual commands are given below in some detail. In each case, the operator must wait for the prompt character (-) before entering the command. If control is passed to a program under test, and the program has not reached a breakpoint, control can be returned to DDT by executing a RST 7 from the front panel (note that the rubout key should be used instead if the program is executing a T or U command). In the explanation of each command, the command letter is shown in some cases with numbers separated by commas, where the numbers are represented by lower case letters. These numbers are always assumed to be in a hexadecimal radix, and from one to four digits in length (longer numbers will be automatically truncated on the right).

Many of the commands operate upon a "CPU state" which corresponds to the program under test. The CPU state holds the registers of the program being debugged, and initially contains zeroes for all registers and flags except for the program counter (P) and stack pointer (S), which default to 100H. The program counter is subsequently set to the starting address given in the last record of a HEX file if a file of this form is loaded (see the I and R commands).

1. The A (Assemble) Command. DDT allows inline assembly language to be inserted into the current memory image using the A command which takes the form

```
As
```

where s is the hexadecimal starting address for the inline assembly. DDT prompts the console with the address of the next instruction to fill, and reads the console, looking for assembly language mnemonics (see the Intel 8080 Assembly Language Reference Card for a list of mnemonics), followed by register references and operands in absolute hexadecimal form. Each sucessive load address is printed before reading the console. The A command terminates when the first empty line is input from the console.

Upon completion of assembly language input, the operator can review the memory segment using the DDT disassembler (see the L command).

Note that the assembler/disassembler portion of DDT can be overlayed by the transient program being tested, in which case the DDT program responds with an error condition when the A and L commands are used (refer to Section IV).

Technology assembler has the side effect in its operation of ignoring the characters after the operand field has been scanned. This causes an ambiguous situation when attempting to be compatible with Intel's language, since arbitrary expressions are allowed in this case. Hence, programs which use this side effect to introduce comments, must be edited to place a ";" before these fields in order to assemble correctly.

The assembly language program is formulated as a seguence of statements of the above form, terminated optionally by an END statement. All statements following the END are ignored by the assembler.

3. FORMING THE OPERAND.

In order to completely describe the operation codes and pseudo operations, it is necessary to first present the form of the operand field, since it is used in nearly all statements. Expressions in the operand field consist of simple operands (labels, constants, and reserved words), combined in properly formed subexpressions by arithmetic and logical operators. The expression computation is carried out by the assembler as the assembly proceeds. Each expression must produce a 16-bit value during the assembly. Further, the number of significant digits in the result must not exceed the intended use. That is, if an expression is to be used in a byte move immediate instruction, then the most significant 8 bits of the expression must be zero. The restrictions on the expression significance is given with the individual instructions.

3.1. Labels.

As discussed above, a label is an identifier which occurs on a particular statement. In general, the label is given a value determined by the type of statement which it precedes. If the label occurs on a statement which generates machine code or reserves memory space (e.g, a MOV instruction, or a DS pseudo operation), then the label is given the value of the program address which it labels. If the label precedes an EQU or SET, then the label is given the value which results from evaluating the operand field. Except for the SET statement, an identifier can label only one statement.

When a label appears in the operand field, its value is substituted by the assembler. This value can then be combined with other operands and operators to form the operand field for a particular instruction.

3.2. Numeric Constants.

A numeric constant is a 16-bit value in one of several bases. The base, called the radix of the constant, is denoted by a trailing radix indicator. The radix indicators are

B	binary constant (base 2)
O	octal constant (base 8)

```
Gs,b,c
G,b
G,b,c
```

The first form starts execution of the program under test at the current value of the program counter in the current machine state, with no breakpoints set (the only way to regain control in DDT is through a RST 7 execution). The current program counter can be viewed by typing an X or XP command. The second form is similar to the first except that the program counter in the current machine state is set to address s before execution begins. The third form is the same as the second, except that program execution stops when address b is encountered (b must be in the area of the program under test). The instruction at location b is not executed when the breakpoint is encountered. The fourth form is identical to the third, except that two breakpoints are specified, one at b and the other at c. Encountering either breakpoint causes execution to stop, and both breakpoints are subsequently cleared. The last two forms take the program counter from the current machine state, and set one and two breakpoints, respectively.

Execution continues from the starting address in real-time to the next breakpoint. That is, there is no intervention between the starting address and the break address by DDT. Thus, if the program under test does not reach a breakpoint, control cannot return to DDT without executing a RST 7 instruction. Upon encountering a breakpoint, DDT stops execution and types

```
*d
```

where d is the stop address. The machine state can be examined at this point using the X (Examine) command. The operator must specify breakpoints which differ from the program counter address at the beginning of the G command. Thus, if the current program counter is 1234H, then the commands

```
G,1234
```

and

```
G400,400
```

both produce an immediate breakpoint, without executing any instructions whatsoever.

5. The I (Input) Command. The I command allows the operator to insert a file name into the default file control block at 5CH (the file control block created by CP/M for transient programs is placed at this location; see the CP/M Interface Guide). The default FCB can be used by the program under test as if it had been passed by the CP/M Console Processor. Note that this file name is also used by DDT for reading additional HEX and COM files. The form of the I command is

```
Ifilename
```

or

of the instruction (e.g, MOV A,B), the value of the instruction (in this case MOV) is the bit pattern of the instruction with zeroes in the optional fields (e.g, MOV produces 40H).

When the symbol "$" occurs in the operand field (not imbedded within identifiers and numeric constants) its value becomes the address of the next instruction to generate, not including the instruction contained withing the current logical line.

3.4. String Constants.

String constants represent sequences of ASCII characters, and are represented by enclosing the characters within apostrophe symbols (´). All strings must be fully contained within the current physical line (thus allowing "!" symbols within strings), and must not exceed 64 characters in length. The apostrophe character itself can be included within a string by representing it as a double apostrophe (the two keystrokes ´´), which becomes a single apostrophe when read by the assembler. In most cases, the string length is restricted to either one or two characters (the DB pseudo operation is an exception), in which case the string becomes an 8 or 16 bit value, respectively. Two character strings become a 16-bit constant, with the second character as the low order byte, and the first character as the high order byte.

The value of a character is its corresponding ASCII code. There is no case translation within strings, and thus both upper and lower case characters can be represented. Note however, that only graphic (printing) ASCII characters are allowed within strings. Valid strings are

```
´A´     ´AB´     ´ab´    ´c´
´´´´    ´a´´´    ´´´´´´  ´´´"´
´Walla Walla Wash.´
´She said ´´Hello´´ to me.´
´I said "Hello" to her.´
```

3.5. Arithmetic and Logical Operators.

The operands described above can be combined in normal algebraic notation using any combination of properly formed operands, operators, and parenthesized expressions. The operators recognized in the operand field are

a + b	unsigned arithmetic sum of a and b
a - b	unsigned arithmetic difference between a and b
+ b	unary plus (produces b)
- b	unary minus (identical to 0 - b)
a * b	unsigned magnitude multiplication of a and b
a / b	unsigned magnitude division of a by b
a MOD b	remainder after a / b
NOT b	logical inverse of b (all 0´s become 1´s, 1´s become 0´s), where b is considered a 16-bit value

assuming the tested program does not destroy the default area at 5CH. Further, any file specified with the filetype "COM" is assumed to contain machine code in pure binary form (created with the LOAD or SAVE command), and all others are assumed to contain machine code in Intel hex format (produced, for example, with the ASM command).

Recall that the command

```
DDT filename.filetype
```

which initiates the DDT program is equivalent to the commands

```
DDT
-Ifilename.filetype
-R
```

Whenever the R command is issued, DDT responds with either the error indicator "?" (file cannot be opened, or a checksum error occurred in a HEX file), or with a load message taking the form

```
NEXT  PC
nnnn pppp
```

where nnnn is the next address following the loaded program, and pppp is the assumed program counter (100H for COM files, or taken from the last record if a HEX file is specified).

9. The S (Set) Command. The S command allows memory locations to be examined and optionally altered. The form of the command is

```
Ss
```

where s is the hexadecimal starting address for examination and alteration of memory. DDT responds with a numeric prompt, giving the memory location, along with the data currently held in the memory location. If the operator types a carriage return, then the data is not altered. If a byte value is typed, then the value is stored at the prompted address. In either case, DDT continues to prompt with successive addresses and values until either a period (.) is typed by the operator, or an invalid input value is detected.

10. The T (Trace) Command. The T command allows selective tracing of program execution for 1 to 65535 program steps. The forms are

```
T
Tn
```

In the first case, the CPU state is displayed, and the next program step is executed. The program terminates immediately, with the termination address

displayed as

*hhhh

where hhhh is the next address to execute. The display address (used in the D command) is set to the value of H and L, and the list address (used in the L command) is set to hhhh. The CPU state at program termination can then be examined using the X command.

The second form of the T command is similar to the first, except that execution is traced for n steps (n is a hexadecimal value) before a program breakpoint is occurs. A breakpoint can be forced in the trace mode by typing a rubout character. The CPU state is displayed before each program step is taken in trace mode. The format of the display is the same as described in the X command.

Note that program tracing is discontinued at the interface to CP/M, and resumes after return from CP/M to the program under test. Thus, CP/M functions which access I/O devices, such as the diskette drive, run in real-time, avoiding I/O timing problems. Programs running in trace mode execute approximately 500 times slower than real time since DDT gets control after each user instruction is executed. Interrupt processing routines can be traced, but it must be noted that commands which use the breakpoint facility (G, T, and U) accomplish the break using a RST 7 instruction, which means that the tested program cannot use this interrupt location. Further, the trace mode always runs the tested program with interrupts enabled, which may cause problems if asynchronous interrupts are received during tracing.

Note also that the operator should use the rubout key to get control back to DDT during trace, rather than executing a RST 7, in order to ensure that the trace for the current instruction is completed before interruption.

11. The U (Untrace) Command. The U command is identical to the T command except that intermediate program steps are not displayed. The untrace mode allows from 1 to 65535 (0FFFFH) steps to be executed in monitored mode, and is used principally to retain control of an executing program while it reaches steady state conditions. All conditions of the T command apply to the U command.

12. The X (Examine) Command. The X command allows selective display and alteration of the current CPU state for the program under test. The forms are

X
Xr

where r is one of the 8080 CPU registers

C	Carry Flag	(0/1)
Z	Zero Flag	(0/1)

M	Minus Flag	(0/1)
E	Even Parity Flag	(0/1)
I	Interdigit Carry	(0/1)
A	Accumulator	(0-FF)
B	BC register pair	(0-FFFF)
D	DE register pair	(0-FFFF)
H	HL register pair	(0-FFFF)
S	Stack Pointer	(0-FFFF)
P	Program Counter	(0-FFFF)

In the first case, the CPU register state is displayed in the format

```
CfZfMfEfIf A=bb B=dddd D=dddd H=dddd S=dddd P=dddd inst
```

where f is a 0 or 1 flag value, bb is a byte value, and dddd is a double byte quantity corresponding to the register pair. The "inst" field contains the disassembled instruction which occurs at the location addressed by the CPU state's program counter.

The second form allows display and optional alteration of register values, where r is one of the registers given above (C, Z, M, E, I, A, B, D, H, S, or P). In each case, the flag or register value is first displayed at the console. The DDT program then accepts input from the console. If a carriage return is typed, then the flag or register value is not altered. If a value in the proper range is typed, then the flag or register value is altered. Note that BC, DE, and HL are displayed as register pairs. Thus, the operator types the entire register pair when B, C, or the BC pair is altered.

III. IMPLEMENTATION NOTES.

The organization of DDT allows certain non-essential portions to be overlayed in order to gain a larger transient program area for debugging large programs. The DDT program consists of two parts: the DDT nucleus and the assembler/disassembler module. The DDT nucleus is loaded over the Console Command Processor, and, although loaded with the DDT nucleus, the assembler/disassembler is overlayable unless used to assemble or disassemble.

In particular, the BDOS address at location 6H (address field of the JMP instruction at location 5H) is modified by DDT to address the base location of the DDT nucleus which, in turn, contains a JMP instruction to the BDOS. Thus, programs which use this address field to size memory see the logical end of memory at the base of the DDT nucleus rather than the base of the BDOS.

The assembler/disassembler module resides directly below the DDT nucleus in the transient program area. If the A, L, T, or X commands are used during the debugging process then the DDT program again alters the address field at 6H to include this module, thus further reducing the logical end of memory. If a program loads beyond the beginning of the assembler/disassembler module, the A and L commands are lost (their use produces a "?" in response), and the

trace and display (T and X) commands list the "inst" field of the display in hexadecimal, rather than as a decoded instruction.

IV. AN EXAMPLE.

The following example shows an edit, assemble, and debug for a simple program which reads a set of data values and determines the largest value in the set. The largest value is taken from the vector, and stored into "LARGE" at the termination of the program

```
ED SCAN.ASM
*I
     ORG     100H    ;START OF TRANSIENT AREA
     MVI     B,LEN   ;LENGTH OF VECTOR TO SCAN
     MVI     C,0     ;LARGER-RST VALUE SO FAR
LOOP:LXI     H,VECT  ;BASE OF VECTOR
LOOP:MOV     A,M     ;GET VALUE
     SUB     C       ;LARGER VALUE IN C?
     JNC     NFOUND  ;JUMP IF LARGER VALUE NOT FOUND
;    NEW LARGEST VALUE, STORE IT TO C
     MOV     C,A
NFOUND:INX   H       ;TO NEXT ELEMENT
     DCR     B       ;MORE TO SCAN?
     JNZ     LOOP    ;FOR ANOTHER
;
;    END OF SCAN, STORE C
     MOV     A,C     ;GET LARGEST VALUE
     STA     LARGE
     JMP     0       ;REBOOT
;
;    TEST DATA
VECT:DB      2,0,4,3,5,6,1,5
LEN  EQU     $-VECT  ;LENGTH
LARGE:DS     1       ;LARGEST VALUE ON EXIT
     END
*BOP
     ORG     100H    ;START OF TRANSIENT AREA
     MVI     B,LEN   ;LENGTH OF VECTOR TO SCAN
     MVI     C,0     ;LARGEST VALUE SO FAR
     LXI     H,VECT  ;BASE OF VECTOR
LOOP:MOV     A,M     ;GET VALUE
     SUB     C       ;LARGER VALUE IN C?
     JNC     NFOUND  ;JUMP IF LARGER VALUE NOT FOUND
;    NEW LARGEST VALUE, STORE IT TO C
     MOV     C,A
NFOUND:INX   H       ;TO NEXT ELEMENT
     DCR     B       ;MORE TO SCAN?
     JNZ     LOOP    ;FOR ANOTHER
```

Create Source Program - underlined characters typed by programmer. "↵" represents carriage return.

```
;       END OF SCAN, STORE C
        MOV     A,C         ;GET LARGEST VALUE
        STA     LARGE
        JMP     0           ;REBOOT
;
;       TEST DATA
VECT:   DB      2,0,4,3,5,6,1,5
LEN     EQU     $-VECT      ;LENGTH
LARGE:  DS      1           ;LARGEST VALUE ON EXIT
        END
*E↵     ← End of Edit
```

```
ASM SCAN↵   Start Assembler
CP/M ASSEMBLER - VER 1.0

0122
002H USE FACTOR
END OF ASSEMBLY     Assembly Complete - Look at Program Listing

TYPE SCAN.PRN↵
```

Code Address ↓ Machine Code ↓ { Source Program

```
0100            ORG     100H        ;START OF TRANSIENT AREA
0100 0608       MVI     B,LEN       ;LENGTH OF VECTOR TO SCAN
0102 0E00       MVI     C,0         ;LARGEST VALUE SO FAR
0104 211901     LXI     H,VECT      ;BASE OF VECTOR
0107 7E     LOOP:   MOV     A,M         ;GET VALUE
0108 91         SUB     C           ;LARGER VALUE IN C?
0109 D20D01     JNC     NFOUND      ;JUMP IF LARGER VALUE NOT FOUND
                ;       NEW LARGEST VALUE, STORE IT TO C
010C 4F         MOV     C,A
010D 23     NFOUND: INX     H           ;TO NEXT ELEMENT
010E 05         DCR     B           ;MORE TO SCAN?
010F C20701     JNZ     LOOP        ;FOR ANOTHER
                ;
                ;       END OF SCAN, STORE C
0112 79         MOV     A,C         ;GET LARGEST VALUE
0113 322101     STA     LARGE
0116 C30000     JMP     0           ;REBOOT
    Code/data listing ;
    truncated →   ;       TEST DATA
0119 0200040305VECT:    DB      2,0,4,3,5,6,1,5
0008 =          LEN     EQU     $-VECT      ;LENGTH
0121 Value of   LARGE:  DS      1           ;LARGEST VALUE ON EXIT
0122 Equate             END
A>
```

```
DDT SCAN.HEX↵
```
Start Debugger using hex format machine code

```
16K DDT VER 1.0
NEXT  PC
0121 0000
-X↵
```
last load address +1

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0000 OUT  7F
-XP↵
```
next instruction to execute at PC=0

Examine registers before debug run

```
P=0000 100↵
```
Change PC to 100

```
-X↵
```
Look at registers again

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI  B,08
-L100↵
```
PC changed.

Next instruction to execute at PC=100

```
0100  MVI  B,08
0102  MVI  C,00
0104  LXI  H,0119
0107  MOV  A,M
0108  SUB  C
0109  JNC  010D
010C  MOV  C,A
010D  INX  H
010E  DCR  B
010F  JNZ  0107
0112  MOV  A,C
-L↵
```
Disassembled Machine Code at 100H (See Source Listing for comparison)

```
0113  STA  0121
0116  JMP  0000
0119  STAX B
011A  NOP
011B  INR  B
011C  INX  B
011D  DCR  B
011E  MVI  B,01
0120  DCR  B
0121  LXI  D,2200
0124  LXI  H,0200
```
A little more machine code (note that Program ends at location 116 with a JMP to 0000)

```
-A116↵
0116  RST 7↵
0117↵
```
enter inline assembly mode to change the JMP to 0000 into a RST 7, which will cause the program under test to return to DDT if 116H is ever executed.

(single carriage return stops assemble mode)

```
-L113↵
```
List Code at 113H to check that RST 7 was properly inserted

```
0113  STA  0121
0116  RST  07
```
In place of JMP

```
0117  NOP
0118  NOP
0119  STAX B
011A  NOP
011B  INR  B
011C  INX  B
-
```

-X→ Look at registers

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI  B,08
```

-I→ Execute Program for one step. initial CPU state, before ↓ is executed

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI  B,08*0102
```

-I→ Trace one step again (note 08H in B) automatic breakpoint ↓

```
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI  C,00*0104
```

-I→ Trace again (Register C is cleared)

```
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0104 LXI  H,0119*0107
```

-T3→ Trace three steps

```
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV  A,M
C0Z0M0E0I0 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB  C
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JNC  010D*010D
```

-D119→ Display memory starting at 119H. automatic breakpoint at 10DH

Program data

```
0119 02 00 04 03 05 06 01
0120 05 11 00 22 21 00 02 7E EB 77 13 23 EB 0B 78 B1  ...."!...^.W.#...x
0130 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 00 00  .'...)..........
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0150 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0160 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0170 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0180 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
0190 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
01C0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00  ................
```

Lowercase x

Data is displayed in ASCII with a "." in the position of non-graphic characters

-X→ Current CPU state ↓

```
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX  H
```

-T5→ Trace 5 steps from current CPU state

```
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX  H
C0Z0M0E0I1 A=02 B=0800 D=0000 H=011A S=0100 P=010E DCR  B
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ  0107
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV  A,M
C0Z0M0E0I1 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB  C*0109
```

Automatic Breakpoint

-U5→ Trace without listing intermediate states

```
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC  010D*0108
```

-X→ CPU state at end of U5

```
C0Z0M0E1I1 A=04 B=0600 D=0000 H=011B S=0100 P=0108 SUB  C
```

```
-G,                 Run Program from current PC until completion (in real-time)
*0116               breakpoint at 116H, caused by executing RST 7 in machine code
-X,                 CPU state at end of Program
C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0116 RST  07
-XP,                examine and change Program counter
P=0116 100,

-X,
C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI  B,08
-T10,               Trace 10 (hexadecimal) steps
C0Z1M0E1I1 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI  B,08
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0121 S=0100 P=0102 MVI  C,00
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0121 S=0100 P=0104 LXI  H,0119
C0Z1M0E1I1 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV  A,M
C0Z1M0E1I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB  C
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JNC  010D
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX  H
C0Z0M0E0I1 A=02 B=0800 D=0000 H=011A S=0100 P=010E DCR  B
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ  0107
C0Z0M0E0I1 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV  A,M
C0Z0M0E0I1 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB  C
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC  010D
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011A S=0100 P=010D INX  H
C0Z1M0E1I1 A=00 B=0700 D=0000 H=011B S=0100 P=010E DCR  B
C0Z0M0E1I1 A=00 B=0600 D=0000 H=011B S=0100 P=010F JNZ  0107
C0Z0M0E1I1 A=00 B=0600 D=0000 H=011B S=0100 P=0107 MOV  A,M*0108
-A109,              Insert a "hot patch" into
0109  JC 10D,       the machine code
                    to change the
010C,               JNC to JC
-G0,                Stop DDT so that a version of
                    the patched Program can be saved
SAVE 1 SCAN.COM,    Program resides on first page, so save 1 page.
A>DDT SCAN.COM,     Restart DDT with the saved memory image to continue testing
16K DDT VER 1.0
NEXT  PC
0200 0100
-L100,              List some code
0100  MVI  B,08
0102  MVI  C,00
0104  LXI  H,0119
0107  MOV  A,M
0108  SUB  C
0109  JC   010D
```

first data element

current largest value

subtract for comparison A<C

Program should have moved the value from A into C since A>C. Since this code was not executed, it appears that the JNC should have been a JC instruction

Previous patch is present in X.COM

```
010C   MOV   C,A
010D   INX   H
010E   DCR   B
010F   JNZ   0107
0112   MOV   A,C
-XP,

P=0100,

-T10,
```

Trace to see how patched version operates

Data is moved from A to C

```
C0Z0M0E0I0 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI   B,08
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI   C,00
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0000 S=0100 P=0104 LXI   H,0119
C0Z0M0E0I0 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV   A,M
C0Z0M0E0I0 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB   C
C0Z0M0E0I1 A=02 B=0806 D=0000 H=0119 S=0100 P=0109 JC    010D
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV   C,A
C0Z0M0E0I1 A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX   H
C0Z0M0E0I1 A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR   B
C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ   0107
C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV   A,M
C0Z0M0E0I1 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB   C
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC    010D
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX   H
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011B S=0100 P=010E DCR   B
C1Z0M0E1I1 A=FE B=0602 D=0000 H=011B S=0100 P=010F JNZ   0107*0107
-X,
```

breakpoint after 16 steps

```
C1Z0M0E1I1 A=FE B=0602 D=0000 H=011B S=0100 P=0107 MOV   A,M
-G,108,
```

Run from current PC and breakpoint at 108H

```
*0108
-X,
```

next data item

```
C1Z0M0E1I1 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB   C
-T,
```

Single Step for a few cycles

```
C1Z0M0E1I1 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB   C*0109
-T,

C0Z0M0E0I1 A=02 B=0602 D=0000 H=011B S=0100 P=0109 JC    010D*010C
-X,

C0Z0M0E0I1 A=02 B=0602 D=0000 H=011B S=0100 P=010C MOV   C,A
-G,
```

Run to completion

```
*0116
-X,

C0Z1M0E1I1 A=03 B=0003 D=0000 H=0121 S=0100 P=0116 RST   07
-S121,
```

look at the value of "LARGE"

```
0121 03,
```

Wrong value!

```
0122 00
0123 22
0124 21
0125 00
0126 02
0127 7E .
-L100
0100  MVI   B,08
0102  MVI   C,00
0104  LXI   H,0119
0107  MOV   A,M
0108  SUB   C
0109  JC    010D
010C  MOV   C,A
010D  INX   H
010E  DCR   B
010F  JNZ   0107
0112  MOV   A,C
-L
0113  STA   0121
0116  RST   07
0117  NOP
0118  NOP
0119  STAX  B
011A  NOP
011B  INR   B
011C  INX   B
011D  DCR   B
011E  MVI   B,01
0120  DCR   B
-XP
P=0116 100
-T
C0Z1M0E1I1 A=03 B=0003 D=0000 H=0121 S=0100 P=0100 MVI   B,03*0102
-T
C0Z1M0E1I1 A=03 B=0803 D=0000 H=0121 S=0100 P=0102 MVI   C,00*0104
-T
C0Z1M0E1I1 A=03 B=0800 D=0000 H=0121 S=0100 P=0104 LXI   H,0119*0107
-T
C0Z1M0E1I1 A=03 B=0800 D=0000 H=0119 S=0100 P=0107 MOV   A,M*0108
```

End of the S Command

Review the Code

Reset the PC

Single Step, and watch data values

Count set

largest set

base address of data set

```
-I
C0Z1M0E1I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB  C*0109
-I
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JC   010D*010C
-I
C0Z0M0E0I1 A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV  C,A*010D
-I
C0Z0M0E0I1 A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX  H*010E
-I
C0Z0M0E0I1 A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR  B*010F
-I
C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ  0107*0107
-I
C0Z0M0E0I1 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV  A,M*0108
-I
C0Z0M0E0I1 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB  C*0109
-I
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC   010D*010D
-I
C1Z0M1E0I0 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX  H*010E
-L100

0100  MVI  B,08
0102  MVI  C,00
0104  LXI  H,0119
0107  MOV  A,M
0108  SUB  C
0109  JC   010D
010C  MOV  C,A
010D  INX  H
010E  DCR  B
010F  JNZ  0107
0112  MOV  A,C
-A108

0108  CMP C

0109

-G0
```

first data item brought to A

first data item moved to C correctly

second data item brought to A

Subtract destroys data value which was loaded!!!

This should have been a CMP so that register A would not be destroyed.

Hot patch at 108H changes SUB to CMP

stop DDT for SAVE

```
SAVE 1 SCAN.COM↵
```
Save memory image

```
A>DDT SCAN.COM↵
```
Restart DDT

```
16K DDT VER 1.0
NEXT  PC
0200 0100
-XP↵

P=0100↵

-L116↵

0116  RST  07
0117  NOP
0118  NOP
0119  STAX B
011A  NOP
- (rubout)
```
Look at code to see if it was properly Loaded (long typeout aborted with rubout)

```
-G,116↵
```
Run from 100H to completion

```
*0116
-XC↵
```
Look at Carry (accidental typo)

```
C1↵

-X↵
```
Look at CPU state

```
C1Z1M0E1I1 A=06 B=0006 D=0000 H=0121 S=0100 P=0116 RST  07
-S121↵
```
Look at "Large" - it appears to be correct.

```
0121 06↵

0122 00↵

0123 22 .↵

-G0↵
```
Stop DDT

```
ED SCAN.ASM↵
```
Re-edit the source program, and make both changes

```
*NSUB↵
*0LT↵
        SUB     C       ;LARGER VALUE IN C?
*SSUB↑ZCMP↑Z0LT↵
        CMP     C       ;LARGER VALUE IN C?
*↵
        JNC     NFOUND  ;JUMP IF LARGER VALUE NOT FOUND
*SNC↑ZC↑Z0LT↵
        JC      NFOUND  ;JUMP IF LARGER VALUE NOT FOUND
*E↵
```
ctl-Z

```
ASM SCAN.AAZ↵
```
Re-assemble, selecting source from disk A
hex to disk A
Print to Z (selects no Print file)

```
CP/M ASSEMBLER - VER 1.0

0122
002H USE FACTOR
END OF ASSEMBLY
```

```
DDT SCAN.HEX↵
```
Re-run debugger to check changes

```
16K DDT VER 1.0
NEXT  PC
0121 0000
-L116↵

0116  JMP  0000
0119  STAX B
011A  NOP
011B  INR  B
- (rubout)

-G100,116↵
```
check to ensure end is still at 116H

Go from beginning with breakpoint at end

```
*0116
-D121↵
```
breakpoint reached

Look at "LARGE"

correct value computed

```
0121 06 00 22 21 00 02 7E EB 77 13 23 EB 0B 78 B1 . . "! . ~.w.#. x.
0130 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 00 00 .'...)..........
0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 ................

-(rubout)
```
aborts long typeout

```
-G0↵
```
stop DDT, debug session complete